NoSQL : Redis
[Core concepts]

Key/Value oriented

- ➔ Map / associative array-like storage
- ➔ Key retrieves the whole data
- ➔ Each field is a key/value pair
- ➔ Examples : Amazon dynamo, project Voldemort and... **Redis**

[“wine1” : [“name”:”Torus”, “color”:”red”, “degre”:13.5]]

[“wine2” : [“name”:”Yquem”, “color”:”white”]]

Redis

- Developed by Salvatore Sanfilippo (2009) ;
- Distributed and replicated in-memory storage system ;
- **Key/Value-oriented** storage ;
- Support persistent storage ;
- https://redis.io ;
- Installation guide : https://redis.io/topics/quickstart ;
- Reference : https://redis.io/commands, http://bl.ocks.org/itamarhaber/e9a87c39467075e2ba76.

Several characteristics

- ➔ Simple queries : get, add, del... ;
- ➔ High performances : O(n), O(log(n)), O(log(log(n))), O(1) ;
- ➔ Indexes on keys (numerical or lexicographical), composite indexes ;
- ➔ Few types managed : string, list, (sorted)set, hash, streams ;
- ➔ Data structure modelling ;
- ➔ Many drivers (Java, Scala, Python, Node.js, Ocaml, Delphi, ...).

Few needed basics admin & tools

- ➔ **redis-server** : server ;
- ➔ **redis-cli** : client console ;
- ➔ **redis-stat** : *top*-like tool for Redis.

Architecture

Schemaless database

- ➔ Server : Master / Slave (cf Clustering) ;
- ➔ Database : 16 numbered databases by default ;
- ➔ Key/value data.

That’s all folks !!

Manage database

- ➔ Use specified database : SELECT *nbr* to switch ;
- ➔ Delete all data from (current|all) DB : FLUSH(DB|ALL) ;
- ➔ Meta-informations : DBSIZE, INFO ;
- ➔ Persist data synchronously : SAVE ;
- ➔ Shutdown server : SHUTDOWN ;

Manage data

- ➔ **No query engine !** (until redis 4.0 search module) ;
- ➔ Simple core add / retrieval ;
- ➔ Performance is the goal : design your data model in accordance ;
- ➔ Choose data structure according to your needs :
  - count elements,
  - retrieve complete complex data structure,
  - retrieve *N* last added elements,
  - retrieve elements within a range ...

Data types

- ➔ String : string, integer or double into binary-safe strings ;
- ➔ List : double-linked list of strings;
- ➔ Hash : hash table of key-value strings
- ➔ Set, sorted set : collection of unique strings. A numerical score defines the order ;
- ➔ Geospatial index : ordered set of triplets {value, longitude, latitude} ;
- ➔ HyperLogLog : structure for counting distinct elements.

Manage keys

- ➔ Keys are binary safe, as string or content of file, max size : 512MB ;
- ➔ Naming convention : multi-word separated by “-”, logical sense separated by “:”, e.g “*user:12042:name*”, “*server:03:ip-addr*” ;
- ➔ Test existence : EXISTS *key* ;
- ➔ Temporary Key : EXPIRE *key sec* (countdown… TTL *key*) ;
- ➔ Renaming : RENAME *old-key new-key* ;
- ➔ Query keys according to a pattern : KEYS *pattern*.

Manage string values

- ➔ Add string value : SET *key value* [NX|XX], MSET *k1 v1 k2 v2* ...;
- ➔ Append value to key : APPEND *key value* ;
- ➔ Retrieve string : GET *key* / GETRANGE *key start-val end-val* ;
- ➔ Delete data : DEL *key* | UNLINK *key* ;
- ➔ Modify values INCR[BY]/DECR[BY] *key* [*nb*].

Manage string values

```
> SET user:1:name peio NX
> SETNX user:1:name peio
> MSET user:1:comp eisti user:1:dept CS user:1:year 10
> SCAN 0
> GET user:1:name
> MGET user:1:name user:1:dept
> SET user:2:name toto XX
> APPEND user:1:name “ loubière”
> GET user:1:name
> GETRANGE user:1:name 0 3
> GETRANGE user:1:name -5 -1
> DEL user:1:name
> GET user:1:name
> DECR user:1:year
> GET user:1:year
> INCRBY user:1:year 3
> GET user:1:year
```

Manage hashes

- ➔ Hashtable, with several key-values pairs ;
- ➔ Values are atomic (no embedded hash nor list nor set) ;
- ➔ Embed complex structure : store key ;
- ➔ Add : H[M]SET *hashkey k1 v1* [*k2 v2…*] ;
- ➔ Retrieve : H[M]GET *hashkey k1* [*k2 k3…*], HGETALL *hashkey* ;
- ➔ Delete : HDEL *hashkey k1*, DEL | UNLINK *hashkey* ;
- ➔ Others : HEXISTS, HKEYS, HVALS, HINCRBY…

Manage hashes

```
> HSET user:01 user:first-name peio
> HSETNX user:01 user:last-name loubiere
> HMSET user:01 user:comp eisti user:dept CS
> HKEYS user:01
> HVALS user:01
> HGET user:01 user:name
> HGETALL user:01
> HMGET user:01 user:name user:dept
> HDEL user:01 user:comp user:dept
> HEXISTS user:01 user:comp
```

Manage lists

- ➔ Double-linked list, often used as a queue ;
- ➔ Avoid insert/remove directly at a position ;
- ➔ Can limit size : capped lists ;
- ➔ Embed complex structure : store key ;
- ➔ Add : LPUSH[X] | RPUSH[X] *list v1* [*v2*...], (LINSERT, LSET) ;
- ➔ Retrieve : LINDEX *list index* ;
- ➔ Delete : LPOP | RPOP *list*, LREM, DEL | UNLINK *list*.

Manage lists

- ➔ Remove element with LREM *list cpt val* :
  - *cpt* > (<) 0 : remove *cpt* first (last) *val* from *list,*
  - cpt = 0 : remove all *val* from *list* ;
- ➔ Extract sub-list : LRANGE *list start stop* ;
- ➔ Capped list : LTRIM *list start stop* ;
- ➔ Pop element with timeout if list is empty : B(L|R)POP *l1 [l2 l3...] sec* ;
- ➔ Sorting : SORT *list* [BY *pattern*] [GET *pattern*] [ALPHA] [ASC|DESC] [STORE *list-dest*] (cf SORT).

Manage lists

```
> LPUSH list:01 2 3 4 5 6
> LRANGE list:01 0 -1
> RPOP list:01
> LRANGE list:01 0 -1
> LPOP list:01
> LINDEX list:01 1
> LTRIM list:01 0 1
> BRPOP list:02 list:01 3
> UNLINK list:01
> BRPOP list:02 list:01 0
> LPUSH list:01 2 3 4
> SORT list:01 DESC
> LPUSH list:01 A b D a c B C
> SORT list:01 DESC
> SORT list:01 ALPHA DESC
```

Manage sets

- ➔ Unordered collection of unique strings ;
- ➔ Ensemble operators (union, intersection, difference) ;
- ➔ Constant retrieving time ;
- ➔ Add : SADD *key v1* [*v2 v3* ...] ;
- ➔ Retrieve : SMEMBERS *key, SSCAN (cf SCAN)* ;
- ➔ Delete : SPOP *key* [*nb*], SREM *key* val, SMOVE *k1 k2 val*, UNLINK *key*.

Manage sets

- ➔ Union : SUNION[STORE *set-res*] *k1* [*k2 k3*...] ;
- ➔ Intersection : SINTER[STORE *set-res*] *k1* [*k2 k3*...] ;
- ➔ Difference : SDIFF[STORE *set-res*] *k1* [*k2 k3*...] ;
- ➔ Get random element : SRANDMEMBER *key*.

Manage sets

```
> SADD set:01 2 3 4 5 6 7 8 9
> SADD set:01 2
> SMEMBERS set:01
> SRANDMEMBER set:01
> SREM set:01 4
> SPOP set:01
> SPOP set:01 2
> SADD set:02 3 5 7 9
> SMOVE set:01 set:02 6
> SMEMBERS set:02
> SADD set:03 8 12 7 42
> SINTER set:01 set:02 set:03
> SUNIONSTORE set:union set:01 set:02 set:03
> SDIFF set:01 set:02 set:03
> SCARD set:union
```

Manage sorted sets

- ➔ Most powerful collection ;
- ➔ Nearly same functionalities as sets but starting with Z...
- ➔ ... but many more ! ;
- ➔ Each element is sorted by a numerical (double) score ;
- ➔ Add : ZADD *key score1 v1* [*score2 v2* ...] ;
- ➔ Retrieve : ZRANGE, ZSCORE, ZRANK, ZSCAN (cf SCAN) ;
- ➔ Delete : ZREM *key v1* [*v2* ...], [B]ZPOPMIN k1 [k2 ...] [time].

Manage sorted sets

- ➔ Union : ZUNIONSTORE *set-res nb k1* [*k2*...] [AGGREGATE SUM|MIN|MAX] ;
- ➔ Intersection : ZINTERSTORE *set-res nb k1* [*k2 k3*...] [AGGREGATE SUM|MIN|MAX] ;
- ➔ Extract : ZRANGE[BYSCORE|BYLEX] *key min max* [WITHSCORES] ;
- ➔ Reverse : ZREVRANGE[BYSCORE|BYLEX] *key max min* [WITHSCORES].

Manage sorted sets

```
> ZADD sset:1 21 toto 17 titi
> ZADD sset:1 9 tata
> ZSCORE sset:1 toto
> ZCARD sset:1
> ZCOUNT sset:1 9 (17
> ZINCRBY sset:1 2 tata
> ZRANGE sset:1 0 -1 WITHSCORES
> ZLEXCOUNT sset:1 te [to //!!!!
> ZRANGEBYSCORE sset:1 8 20 WITHSCORES
> ZREVRANGEBYSCORE sset:1 20 8 WITHSCORES
> ZADD sset:2 33 toto 5 tutu
> ZUNIONSTORE sset:3 2 sset:1 sset:2 AGGREGATE MAX
> ZSCORE sset:1 tata
> ZRANK sset:1 tata
```

Manage hyperloglog

- ➔ Used to count approximatively **unique** elements ;
- ➔ Provide an answer with an error of 0.81% ;
- ➔ Add : PFADD *key v1* [*v2* ...] ;
- ➔ Merge sets : PFMERGE *key-res key1* [*key2*...] ;
- ➔ Let's count hyperloglog-style : PFCOUNT *key1* [*key2*...] ;

Manage hyperloglog

```
> PFADD hll:1 2 3 4 5 6 7 8 9 2 7 21 3
> PFADD hll:2 3 42 15 6 64 7 31 23
> PFCOUNT hll:1 hll:2
> PFMERGE hll:3 hll:1 hll:2
> PFCOUNT hll:3
```

Pub/Sub

- (Un)Subscribe to specific channels or using patterns : [P](UN)SUBSCRIBE *ch1* [*ch2 ch3...*] ;
- Subscribers must be up to receive messages. If not, messages will be lost ;
- Post message to given channel : PUBLISH *ch mess* ;
- Get informations on channels : PUBSUB.

Pub/Sub

```
*** client 1 ***
> PSUBSCRIBE news.*

*** client 2 ***
> SUBSCRIBE news.sport

*** sender ***
> PUBLISH news.sport “BO: 12 - AB: 18”
> PUBLISH news.music “Lemmy Kilmister died at 70.“
> PUBSUB CHANNELS
> PUBSUB CHANNELS news.*
> PUBSUB NUMSUB news.sport
> PUBSUB NUMSUB news.music
> PUBSUB NUMPAT
```

Manage Streams

- ➔ Append-only data structure ;
- ➔ Allow to store structured data ;
- ➔ Basic manipulations same as lists ;
- ➔ Add : XADD *stream-key (id | *) k1 v1* [*k2 v2* ...] ;
- ➔ Returns the ID of the added element (default : *timestamp-nb)* ;
- ➔ Many ways to consume data, by many clients :
    - by id range ;
    - browsing with cursor;
    - only the newest.

Manage Streams

- ➔ Consume by range : X[REV]RANGE *key* (-|id_{min}) (+|id_{max}) [COUNT *nb*] ;
- ➔ Consume :
- ➔ XREAD [COUNT *nb*] [BLOCK *n*] STREAMS *k1* [*k2* ...] (id_{min}1|$) [id2 ...] ;
- ➔ BLOCK *n* : same as BLPOP, BRPOP, ZPOPMIN (n=0 : no timeout) ;
- ➔ $ stands for new entry since we blocked ;

Manage Streams

- ➔ Consumer Group : XGROUP *stream-key group-name* (id_{min} | $) ;
- ➔ Allow to distribute messages (1 per consumer) ;
- ➔ Consumers are added implicitly using XREADGROUP ;
- ➔ Consume :
- ➔ XREADGROUP GROUP group-name consumer-name

[COUNT *nb*] [BLOCK *n*] [NOACK] STREAMS *k1* [*k2* ...] (id_{min}1 | $) [id2 ...]

Manage Streams

```
> XADD tweets * author aaa content “blabla aaa 1”
> XADD tweets * author zzz content “blabla zzz 1”
> XADD tweets * author aaa content “blabla aaa 2”
> XADD tweets * author eee content “blabla eee 1”
> XADD tweets * author rrr content “blabla rrr 1”
> XLEN tweets
> XRANGE tweets - +
> XRANGE tweets - + COUNT 2
> XREAD COUNT 2 STREAMS tweets 0
> XREAD BLOCK 0 STREAMS tweets $
> XADD tweets * author rrr content “blabla rrr 2”
```

Manage Streams

```
> XADD jobs * sender aaa todo job1
> XGROUP CREATE jobs doers $
> XREADGROUP GROUP doers pc1 COUNT 1 BLOCK 0 STREAMS jobs >
> XREADGROUP GROUP doers pc2 COUNT 1 BLOCK 0 STREAMS jobs >
> XREADGROUP GROUP doers pc3 COUNT 1 BLOCK 0 STREAMS jobs >
> XINFO GROUPS jobs
> XADD jobs * sender aaa todo job1
> XADD jobs * sender aaa todo job2
> XADD jobs * sender aaa todo job3
> XPENDING jobs doers
> XACK jobs doers id-retourné
> XPENDING jobs doers
> XPENDING jobs doers - + 4
> XPENDING jobs doers - + 4 pc3
> XADD jobs * sender aaa todo job4
> XADD jobs * sender aaa todo job5
> XREADGROUP GROUP doers pc1 BLOCK 0 STREAMS jobs > //COUNT !
```

Geospatial API

- ➔ Use 2D sphere GPS coordinates : **[longitude,latitude]** ;
- ➔ Stored in a sorted set ;
- ➔ Simple queries : add (GEOADD), get (GEO(POS|HASH)), distance (GEODIST) and within (GEORADIUS[BYMEMBER]) ;
- ➔ Manage different units : {m, km, ft, mi} ;
- ➔ Delete : use of *ZREM*.

Geospatial API

- GEOADD key long lat member [...] ;
- GEO(POS|HASH) key member ;
- GEODIST key member1 member2 [unit] [STORE key] ;
- GEORADIUS key long lat radius [unit] [STORE key];
- GEORADIUSBYMEMBER key member radius [unit] [STORE key].

Geospatial API

```
> GEOADD pa 43.3 -0.3667 pau
> GEOADD pa 43.194413 -0.6107 oloron
> GEOADD pa 43.48756 -0.773928 orthez
> GEOADD pa 43.492949 -1.474841 bayonne
> GEOPOS pa oloron
> GEODIST pa pau bayonne
> GEORADIUS pa 43.2 -0.5 60 km
> GEORADIUSBYMEMBER pa pau 50 km
> ZREM pa orthez
```

Sorting

- ➔ Powerful sorting operation with pattern selection and extraction and paging ;
- ➔ Apply to lists, sets and sorted sets ;
- ➔ SORT *key* [BY *pattern*] [LIMIT *start nb*] [GET *pattern* [GET *pattern* ...]] [ASC|DESC] [ALPHA] [STORE *dest-key*].

Sorting

```
> LPUSH list:02 8 3 6 12 4 9 49
> HSET usr:1 nom toto age 42
> HSET usr:2 nom titi age 12
> HSET usr:3 nom tata age 27
> LPUSH list:03 1 2 3
> SORT list:02 LIMIT 0 3 DESC
> SORT list:03 BY usr:*->age
> SORT list:03 BY usr:*->age GET usr:*->nom STORE list:03:sorted-name
> EXPIRE list:03:sorted-name 5
> LRANGE list:03:sorted-name 0 -1
....
> LRANGE list:03:sorted-name 0 -1
```

Scan

- ➔ Iterate over collections (SCAN, SSCAN, HSCAN, ZSCAN) ;
- ➔ Iterate the current database : SCAN 0 ;
- ➔ Iterate collections : [S|H|Z]SCAN *key cursor* [MATCH *p*] [COUNT *n*] ;
- ➔ first part of the result : next cursor value ;
- ➔ 0 is the termination cursor, no more values to iterate.

Scan

```
> SCAN 0
> SADD set:1 a2 b3 a4 c5 b6 b7 a8 c9 a9 b1 c2 a5 b5
> SSCAN set:01 0
> SSCAN set:01 0 COUNT 3
> SSCAN set:01 0 MATCH a* COUNT 3
```

Transactions

- ➔ Set several commands to be executed in a single operation ;
- ➔ Start queuing commands : MULTI ;
- ➔ Execute group of commands : EXEC ;
- ➔ Cancelling current transaction : DISCARD :
- ➔ Prevent transaction to be executed if some keys are modified by other clients : WATCH *k1* [*k2*...] ;
- ➔ **If failure after EXEC, no rollback**.

Transactions

```
> ZADD sset:1 21 toto 17 titi
> MULTI
> ZADD sset:1 9 tata
> ZINCRBY sset:1 2 tata
> ZINCRBY sset:1 7 toto
> EXEC

> MULTI
> SET hello world
> LPOP hello
> EXEC
```

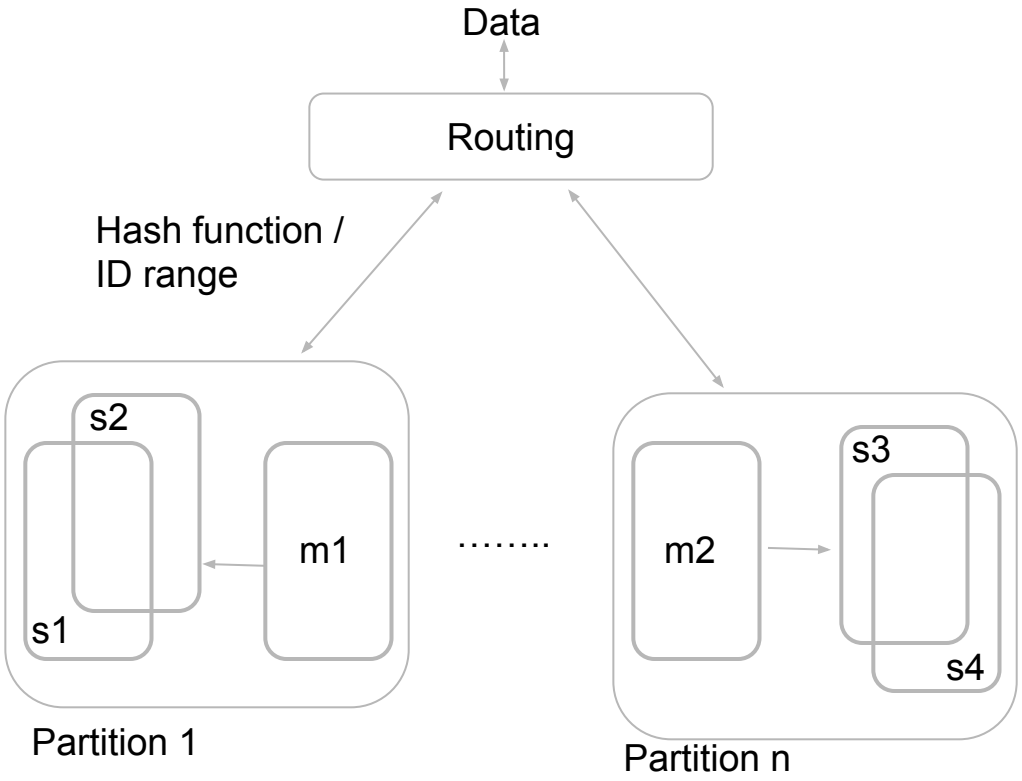
Persistence

- ➔ 2 kinds of persistence : RDB, AOF
- ➔ RDB : compact file, use to back-up data regularly (snapshots), may cause some data loss if a crash occurs;
- ➔ AOF : more powerful, safer, with log and sync options but slower and require bigger files ;
- ➔ Configuration : `redis.conf` file :
  - append-only no : disable persistence
  - save *seconds nb-changes*: periodicity of rdb snapshotting (without : no persistence),
  - dbfilename : name of dump file,
  - dir : path to dump files directory.

Clustering

- ➔ Not very suited for sharding data :
  - multiple keys commands are not available,
  - union and intersection can manage data from the same node;
- ➔ Partitioning according keys, for each object (range/hash);
- ➔ Master-Slaves replication;
- ➔ *Sentinel* software provides **high availability**, monitoring and failover management **for replication**;
- ➔ *Redis cluster* provides **sharding capabilities** with failover management and **replication**;
- ➔ *Redis cluster* provides slaves migration to switch available slave to a master that no longer have one.

Sharding and replication : a little overview



Replication

- ➔ Master/Slaves architecture;
- ➔ The master node replicates data to slaves nodes, asynchronously;
- ➔ Synchronous replication using **`WAIT`** command;
- ➔ Slaves can be used as scalability issue, to execute processes with high complexity;
- ➔ If a slave disconnects, the master will check for availability and then synchronises data (partial or full);
- ➔ If a master disconnects, a slave node is promoted master node.

Replication

- ➔ Replication is managed by a couple (ID,offset) :
  - ○ ID defines the master node
  - ○ offset defines the version of data, manages synchronization;
- ➔ By default, client nodes are read only nodes, they refuse write requests;
- ➔ Master node can be set to accept write queries when at least *n* slave nodes are connected.

Sharding : how to split data ?

- ➔ ID Range for each collection : a set of value is given to each node :
  - data storage not well balanced,
  - **impossible to manage multiple keys**,
  - mapping tables of ranges for each object are needed
- ➔ Hash function (eg. mda, crc16, crc32) on the key and modulo the number of partitions.

Sharding : who cares ?

- Different routing techniques :
  - **The client** : selects the right partition to write/read (not implemented in all languages);
  - **A proxy** : the client node sends the request to a proxy that manages the cluster (eg. Twenproxy, Codis);
  - **Query routing** : the client node sends the query to a random partition node, then it is forwarded to the correct partition node (eg. Redis cluster).

Redis 5.0.1 & others, to go further

- ➔ Modules, introducing :
  - new data types **JSON**,
  - search engine,
  - neural network and machine learning,
  - and many more...
- ➔ Lua scripting ;
- ➔ ...

Thank you for your attention